Performance Analysis of TCP Variants

Jyothi Prasad Nama Mahesh, Srinivasa Nikith Tumati

Abstract – *TCP is widely used in most of today's Internet applications. Many TCP variants were introduced, each improving the performance of TCP in some way in different network conditions. The aim of this paper is to discuss the detailed analysis of some of the TCP variants – Tahoe, Reno, Newreno, Vegas and Sack with the help of Network Simulator–2 (NS-2). It also briefly discusses about the fairness between various TCP variants. All performances are evaluated based on the connection's response to network parameters like CBR, propagation delay, latency, packet drop conditions and so on. Each experiment aims at answering certain statistical questions such as which variant has best throughput, which variant performs best, which are variants fair to each other and so on. Such analysis is immensely in need to be aware of which TCP is better for a specific criterion, wherefrom an appropriate one will be selected in respective network to optimize traffic goal.*

I. INTRODUCTION

Transmission Control Protocol (TCP) is the core protocol of the Internet Protocol suite which provides reliable, connection-oriented, ordered and error-checked end-to-end delivery of streams of octets (bytes) of data by packet-switching between applications running on hosts communicating over an IP network. TCP also provides congestion control and flow control for better performance and reliability. In order to conduct the congestion control mechanism there are few TCP variants introduced and there are continuous enhancements being implemented. The **TCP Tahoe** is the base variant which was introduced with basic congestion control algorithms - slow start, congestion avoidance and fast retransmission. Later a new algorithm was introduced, fast recovery, in **TCP Reno.** Some improvements were introduced to Reno's retransmission algorithm during the fast-recovery phase of and a new variant **TCP Newreno** was introduced. After a lot of research, a new variant **TCP Vegas** was introduced where timeouts and round trip delays are estimated for every packet present in the transmit buffer and such an implementation detects congestion in a proactive manner.

Analysis of these variants is important to determine the best performance of TCP under the influence of these variants for a specific network scenario. This paper contains the discussion of the results of various results of simulation-based experiments to understand the performance of the TCP variants. The focus on the simulation environment is the presence of another variant or the presence of a CBR flow.

II. METHODOLOGY

The experiment setup was simulated using *Tool Command Language (TCL)* that ran *Network Simulator 2 (NS-2)* for the network topology and the analysis was performed using *"Microsoft Excel"* to plot the graphs and evaluate its performance. This NS2 tool creates trace files (.tr) of all the transmission of data in the network during the experiment which helps in understanding the behaviour of the packets in the network at different scenarios. These trace files contain information like type of even occurred, time at which the event occurred, from and to node, packet type, flow id, source and destination address with port numbers and sequence numbers. Hence, we have used python scripts to parse the trace file to extract the required data and stored in a new file to plot the graph for the analysis of various experiments.

The network topology used in the experiment has 6 nodes and 5 duplex links connecting the nodes to form a network as shown in fig (1). The links that connect the nodes have bandwidth of 10Mbps and a delay of 10ms and the link between node 2 and node 3 has a queue limit of 10. All links implement a DropTail queueing technique, unless required otherwise. The nodes implement TCP or UDP in the transport layer and FTP or CBR in the application layer respectively. Senders implement one of the different TCP variants under discussion and receivers implement a sink or a null.

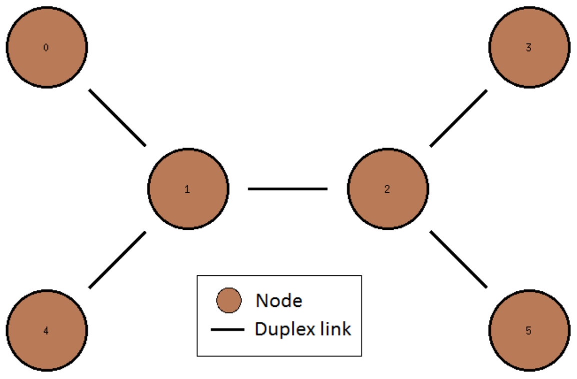

Fig (1) – Network topology

The application layer protocols' attributes can be set as required, such as packet size, bit rate (for CBR traffic) to test the performance under different circumstances.

1. TCP Performance under Congestion

To start off with the analysis of the TCP variants, the topology is set-up as described above. A transport layer agent is attached to each node i.e., N1–TCP source, N4–TCP sink, N2–UDP source and N3–null. All UDP agents run CBR traffic and TCP agents run FTP traffic in the application layer protocols. The aim of this experiment is to analyse the performance of TCP variants in the influence of the CBR traffic. The droptail queuing algorithm is set to a queue limit 10.

The experiment is run for 10s and the results are sampled at various CBR values ranging from 0.1Mbps to 10Mbps at increments of 0.1. Thus we get a sample of O(100) to perform the analysis. The test is performed till the bottleneck capacity is reached (10Mbps) so that there is a gradual increase in the congestion. This experiment is run for all the 4 TCP variants under study and a detailed analysis is done with the help of graphs plotted of the values tabulated. The analysis of the throughput, packet drop rate and latency of the data with respect to the CBR gives a big picture of the performance levels of different TCP variants.

2. Fairness between TCP Variants

Next experiment is performed to understand the influence of one TCP connection on another connection of the same or different type. In this, two TCP flows and one CBR flow are set-up between N1-N4, N5-N6 and N2-N3 respectively. The queue is set to 10 and this uses DropTail queuing algorithm just like in the first experiment. The experiment is run till the bottleneck capacity is reached and it is performed for Reno vs Reno, Newreno vs Reno, Vegas vs Vegas, and Newreno vs Vegas. The recorded values in the .tr file are parsed and graphs are plotted for the analysis of their fairness.

The experiment is run for a total of 10s where in the CBR traffic is started first at 0.5s mark and then the FTP traffics are started at 1.0s mark. The FTP traffics are stopped before the CBR at 9.5s mark and 10.s mark respectively. The trace file will contain data for all the traffics together. Each TCP connection can be differentiated by their source and destination addresses. Based on this, the results are parsed and used to plot the graphs. The distribution of the graphs will give us a clear idea on which variant performs better and are they being fair to each other or not.

3. Influence of Queuing

The last experiment aims at determining the effect of various queueing techniques. There are several queueing techniques each of which tackles some issue to improvise the TCP performance. The most naïve implementation is DropTail technique that just drops the packets that arrive when the queue is full else does nothing. This is a type of passive queue management. Another technique under consideration is RED (Random Early Detection) which is a type of active queue management. It starts to drop packets (or mark with ECN flag) at random when the average queue size is larger than a minimum threshold, and drops all packets when the average queue size is larger than a maximum threshold.

To perform this experiment one TCP (N1-N4) and one UDP (N5-N6) stream is used. The simulation starts with the TCP connection first and the CBR traffic is introduced once the TCP stabilizes (which is around 10s). The performance of two variants, TCP Reno and TCP Sack, are analysed with the estimation of throughput and latency. This experiment is repeated for both the queuing techniques.

III. RESULTS AND ANALYSIS

1. TCP Performance under Congestion

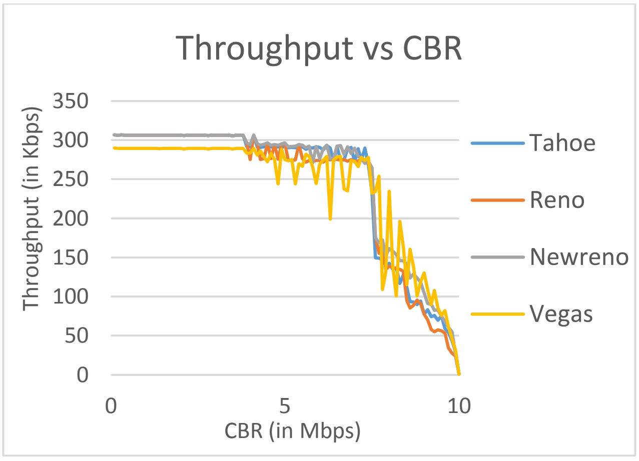

Fig (2) Performance comparison of TCP Variants

To analyse the performance of the variants, a graph with all the throughputs of the variants are plotted against varying CBR values. Fig (2) shows the graph. Initially all variants perform similarly with not much variations. This is because the CBR bandwidth is still not high enough to cause congestion in the links. Once congestion occurs, there is a lot of variance in the throughput of all the TCP connections, Vegas being the most varying with a standard deviation of 84.8 Kbps after congestion occurs. The see-saw pattern that occurs after congestion is due to the additive increase and multiplicative decrease of the *cwnd*. Also, Vegas has a relatively higher average throughput of 240Kbps.

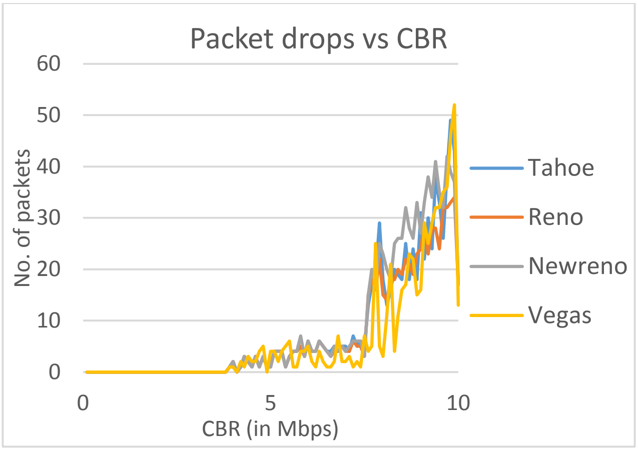

Fig (3) Packet drop comparison of TCP Variants

From the above figure, it seems like Vegas has slightly fewer packet drops. In fact, Vegas has the least average packet drops, equal to 6.4, whereas the other variants ranged from 7-10.3. This is because Vegas uses the RTT (Round Trip Time) in its congestion avoidance algorithm rather than number of dropped packets. After every packet transfer that is ACKed, Vegas calculates a new rtt value from the old and the current sample thus reducing packet drops.

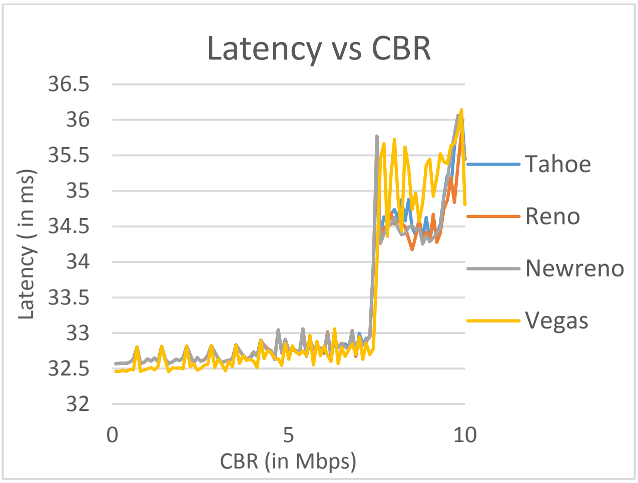

Fig (4) Latency comparison of TCP Variants

From the above figure, we can observe that Vegas has a low latency initially and when congestion builds up, it varies widely. Overall, Vegas has the least average latency of 30.4ms while the others range from 33-35ms. This is because the congestion avoidance algorithm of Vegas samples the rtt value after every transmission and does not depend on a timeout or duplicate ACKs.

From the results of this experiment, we can conclude that Vegas performs the best under congested conditions but not very impressive when the links are not congested. Since the real life conditions converge more towards the former case, we can say that Vegas is a good choice among the four to use in connections with large congestions.

2. Fairness between TCP Variants

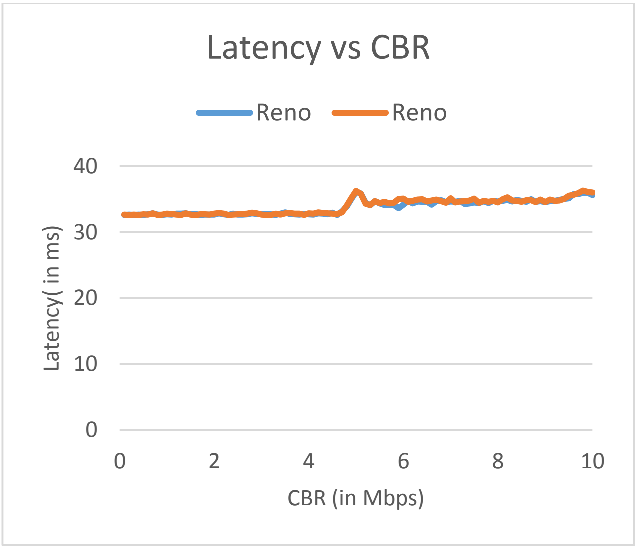

Fig (5) Latency comparison of Reno with Reno

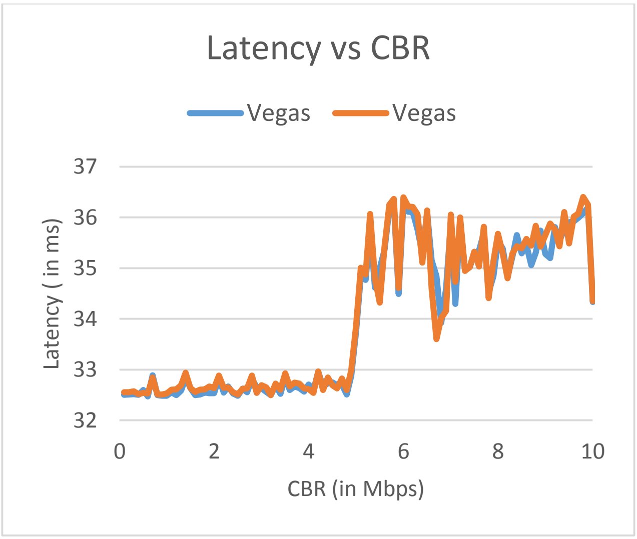

Fig (6) Latency comparison of Vegas with Vegas

Fig (5) and (6) show the latencies of two TCP connections of the same variants (Reno and Vegas, respectively). It is obvious that they must perform similarly because they are of the same type running under same conditions. Hence we don't see much variation between the two variants in each of the graphs above. A test with one variant starting before another showed that the performance of the first connection converged to that of second connection once it started.

The following two graphs show the variance of throughput of two connections running simultaneously, just as before, but with dissimilar variants. Since each variant implements different algorithms, they must perform differently.

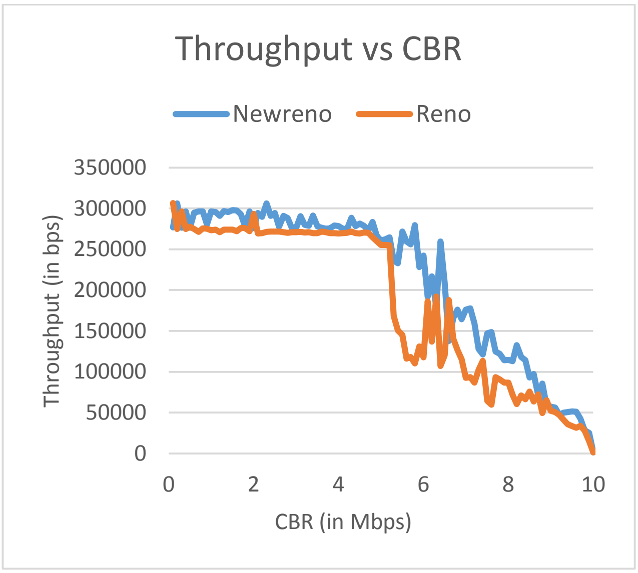

Fig (7) Throughput comparison of Newreno with Reno

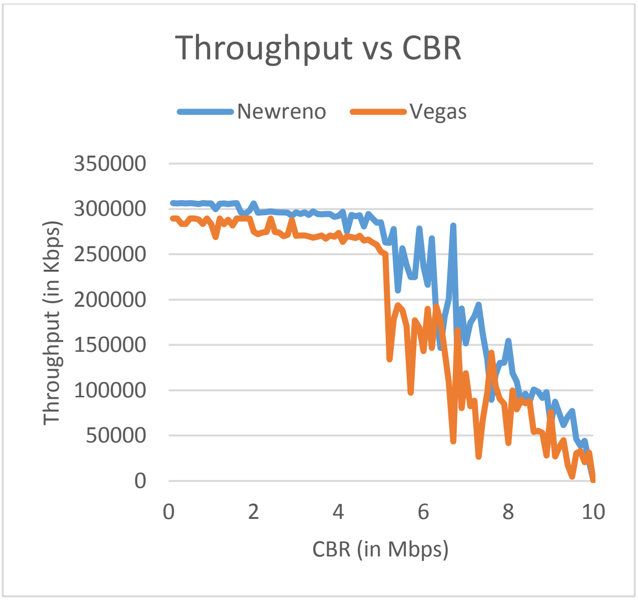

Fig (8) Throughput comparison of Newreno with Vegas

In Fig (7), we see that Newreno has relatively more throughput than Reno with their average throughputs being 214 Kbps and 185 Kbps, which is a significant difference. This is because Reno waits for three dup-ACKs before retransmitting unacked packets whereas Newreno retransmits after every dup-ACK it receives.

In Fig (8), we see that again Newreno consumes higher throughput than Vegas with average throughputs of 222 Kbps and 186 Kbps. This is because Vegas' congestion control algorithm estimates the RTT and adjusts its cwnd accordingly, meanwhile Newreno keeps sending more and more packets as it gets more space in the queue as it does not vary its cwnd based on the congestion, rather it varies it based on unacked packets.

Thus we can conclude that two variants of the same type are fair to each other but different variants are not fair. This has a negative impact when implemented in real world as there will be huge number of connections running simultaneously, each implementing different variant and each connection will not receive fair amount of bandwidth.

3. Influence of Queuing

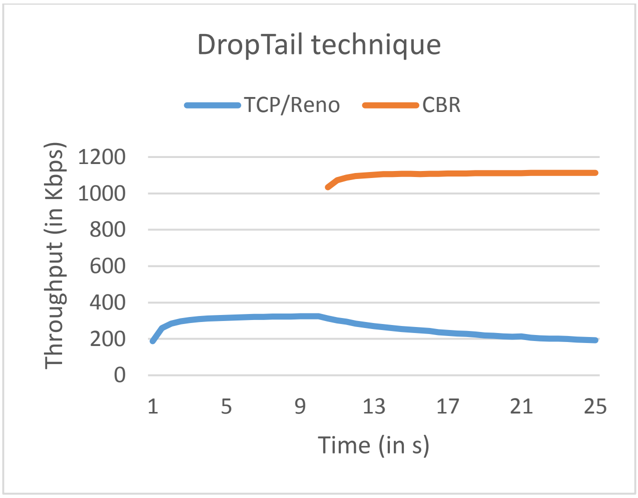

Fig (9) Throughput comparison for DropTail algorithm

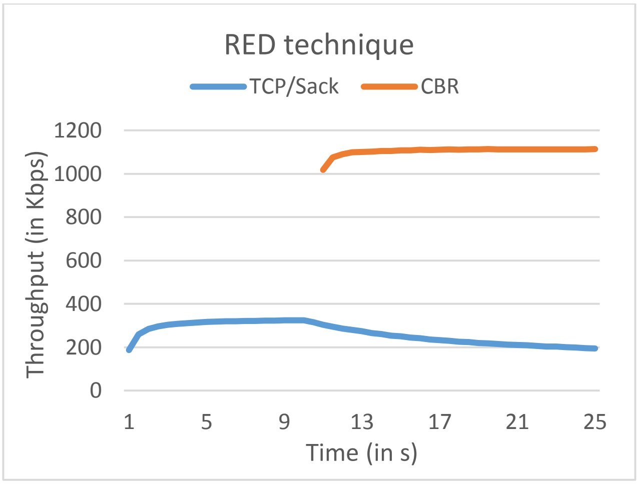

Fig (10) Throughput comparison for RED algorithm

From the plotted graphs we can see that the queuing mechanisms don't provide equal fairness to TCP and UDP flows. The variant of TCP used has no effect on the throughput and both queueing techniques favour CBR traffic more than the FTP traffic here.

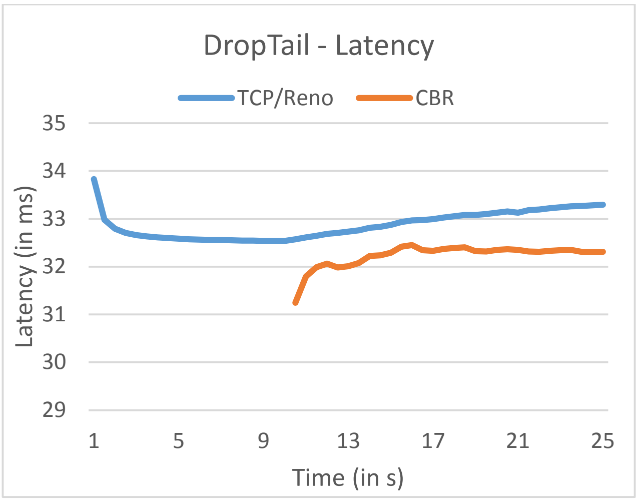

Fig (11) Latency comparison for DropTail algorithm with TCP/Reno and UDP

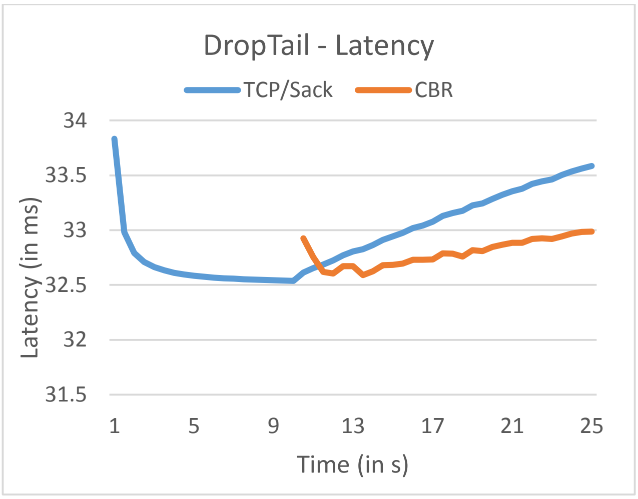

Fig (12) Latency comparison for DropTail algorithm with TCP/Sack and UDP

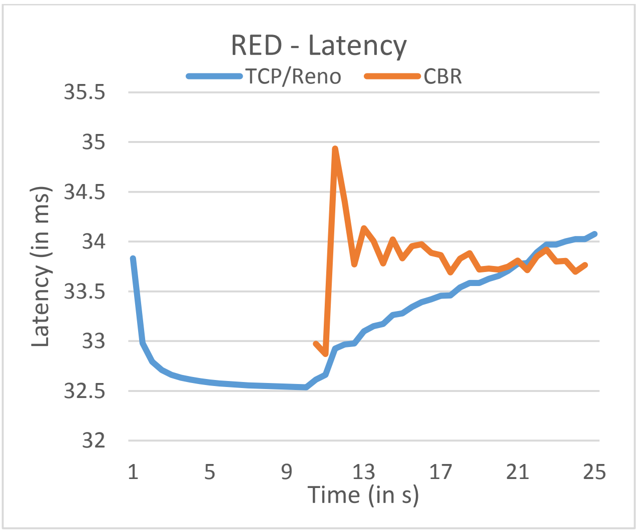

Fig (13) Latency comparison for RED algorithm with TCP/Reno and UDP

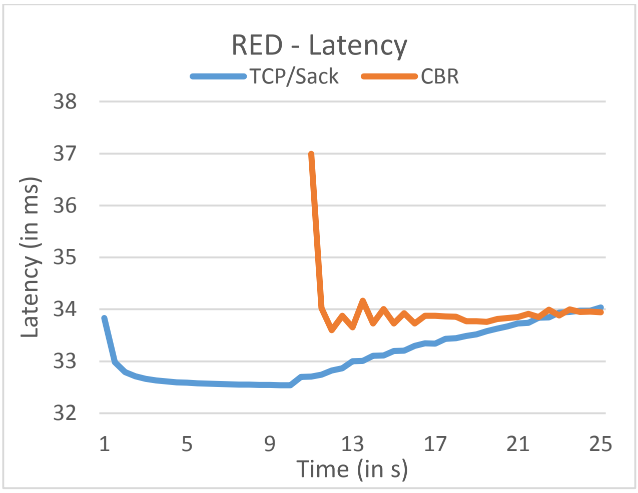

Fig (14) Latency comparison for RED algorithm with TCP/Sack and UDP

We can observe that the end-to-end latency for the flows differ widely between DropTail and RED. This is because the TCP DropTail algorithm gives more preference to bursty traffic but RED is fair to all types of data in the network. So we can say that RED performs better while DropTail does not and it is inclined more towards bursty traffic. All the packets that arrive after the queue is filled are dropped in the DropTail technique but the RED drops the packets randomly based on statistical probability. When the number of queued packets exceeds certain minimum threshold value, it either marks or drops packets randomly to indicate congestion. Thus we can say that the end-to-end latency is more in DropTail algorithm when compared to RED.

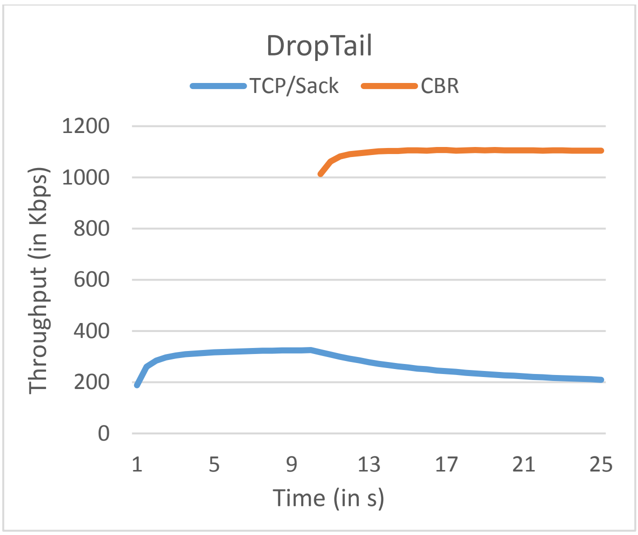

Fig (15) Throughput comparison for DropTail algorithm

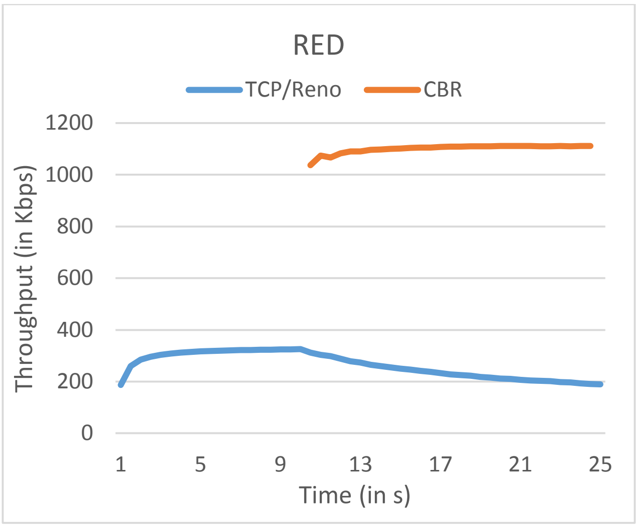

Fig (16) Throughput comparison for RED algorithm

In general, both the queueing techniques provide more throughput to CBR traffic and moreover, once congestion is introduced the throughput of the TCP reduces below its stable value before the contention generated by CBR traffic.

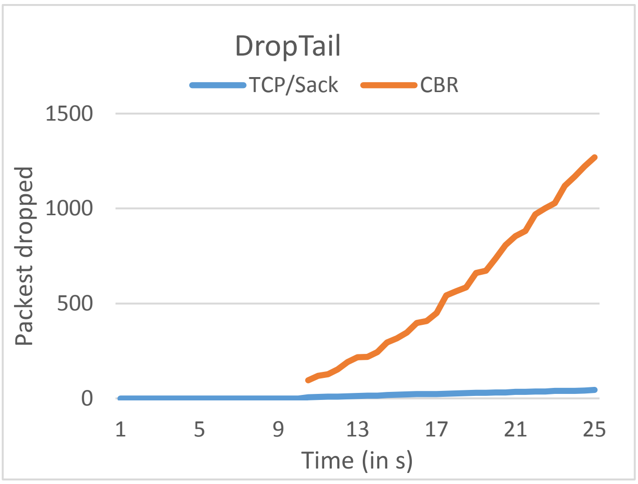

Fig (17) Packet drop comparison for DropTail algorithm with TCP/Sack

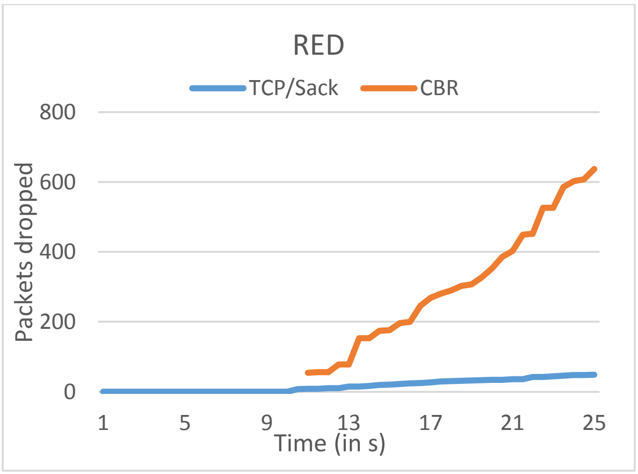

Fig (18) Packet drop comparison for RED algorithm with TCP/Sack

From Fig (17) and (18), we can see that although the distribution of the number of packets dropped look similar, the number of packets dropped by RED is much lesser than that dropped by DropTail for the CBR traffic. But the RED drops more TCP/Sack packets – 841 packets – than DropTail - 757. RED drops an average of 307.75 UDP packets and DropTail drops an average of 588.87 UDP packets. Under the same conditions, RED drops more TCP packets than DropTail and hence we can say that RED is not a good choice while dealing with SACK.

IV. CONCLUSION

This paper briefly describes various variants of TCP by performing analysis of performance of Tahoe, Reno, Newreno and Vegas TCP variants. Key insights from the experiments would be that Vegas has a significant improvement in bandwidth utilization and minimal packet drops. We also learnt that different variants are not fair to each other at all time. One important observation would be that Newreno consumes relatively high bandwidth with respect to bot Reno and Vegas which suggests that although Newreno is an improvement over some TCP variant, it is not fair when implemented in the real world which beats the purpose of it. Lastly, we learnt that RED is more honest to all connections while DropTail is more inclined towards bursty networks and that RED is not a good choice with TCP/SACK

V. REFERENCES

[1] *"TCP and Queue Management"* - cp.literature.agilent.com/litweb/pdf/7873EN.pdf

[2] How to use Statistical tools to interpret data - www.fao.org/docrep/w7295e/w7295e08.htm

[3] *"TCP Performance"* by *Geoff Huston, Telstra* http://www.cisco.com/web/about/ac123/ac147/ac174/ac196/about_cisco_ipj_archive_article09186a00800c8417.html

[4] TCL and NS-2 coding syntax and objects - http://www.isi.edu/nsnam/ns/doc/

[5] *"A Comparative Analysis of TCP Tahoe, Reno, New-Reno, SACK and Vegas*" - http://inst.eecs.berkeley.edu/~ee122/fa05/projects/Project2/SACKRENEVEGAS.pdf

[6] *"Scenario Based Performance Analysis of Variants of TCP using NS2-Simulator "* - http://omicsonline.org/open-access/scenario-based-performance-analysis-of-variants-of-tcp-using-nssimulator-1-0976-4860-1-223-233.pdf